Low-overhead trace collection and profiling on GPU compute kernels

Sébastien Darche <sebastien.darche *at* polymtl.ca>

December 8, 2022

Dorsal - Polytechnique Montréal

Introduction

- GPUs have become ubiquitous in many fields, notably HPC and machine learning
- Multiple programming models have been developped, both low and high level
  - CUDA, HIP, OpenCL
  - SYCL, OpenMP, OpenACC
- GPU programming remains a difficult task

Motivation

- Tooling is maturing
  - ROC-profiler
  - Intel VTune
  - HPCToolkit[1], ...
- Mostly from the point of view of the host
- However, little to no information about what is actually happening on the device

[1] K. Zhou, L. Adhianto, J. Anderson, *et al.*, "Measurement and analysis of gpu-accelerated applications with hpctoolkit," *Parallel Computing*, vol. 108, p. 102 837, 2021.

Shortcomings of current work

- Most tools rely on hardware performance counters and/or PC sampling
- CUDAAdvisor[2] proposes LLVM-based instrumentation of compute kernels
  - little consideration for overhead (costly kernel-wide atomic operations)
  - Overhead ranging from $\sim 10\times$ to $120\times$
- CUDA Flux[3] introduces CFG instrumentation combined with static analysis
  - only one thread is instrumented, does not support divergence
  - Overhead ranging from $\sim 1\times$ to $151\times$ (avg. $13.2\times$)

---

[2] D. Shen, S. L. Song, A. Li, *et al.*, "Cudaadvisor: Llvm-based runtime profiling for modern gpus," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018.

[3] L. Braun and H. Fröning, "Cuda flux: A lightweight instruction profiler for cuda applications," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2019.

Method

We propose a method for instrumenting kernel execution on the GPU with a minimal runtime overhead.

- Relies on a set of LLVM passes for the host and device IR
- Multi-stage performance analysis
  - CFG counters to retrieve the control flow of the program
  - Event collection for precise analysis
  - Optionally, original kernel for timing data
- Knowledge of the control flow allows for pre-allocation of the buffers
- Deterministic execution is ensured by reverting memory

Use cases

- First run (low overhead) can generate accurate custom software performance counters from the CFG counters

Event collection is more costly, but much more insightful :

- Control flow analysis
  - Precise execution sequence to analyze wavefront lifetime, thread divergence
- Precise memory divergence analysis
  - Could point out poor data locality (how and where), which is critical in GPUs
- Extensible data collection framework

In-between heavyweight GPU simulation and lightweight profiling

Quick example

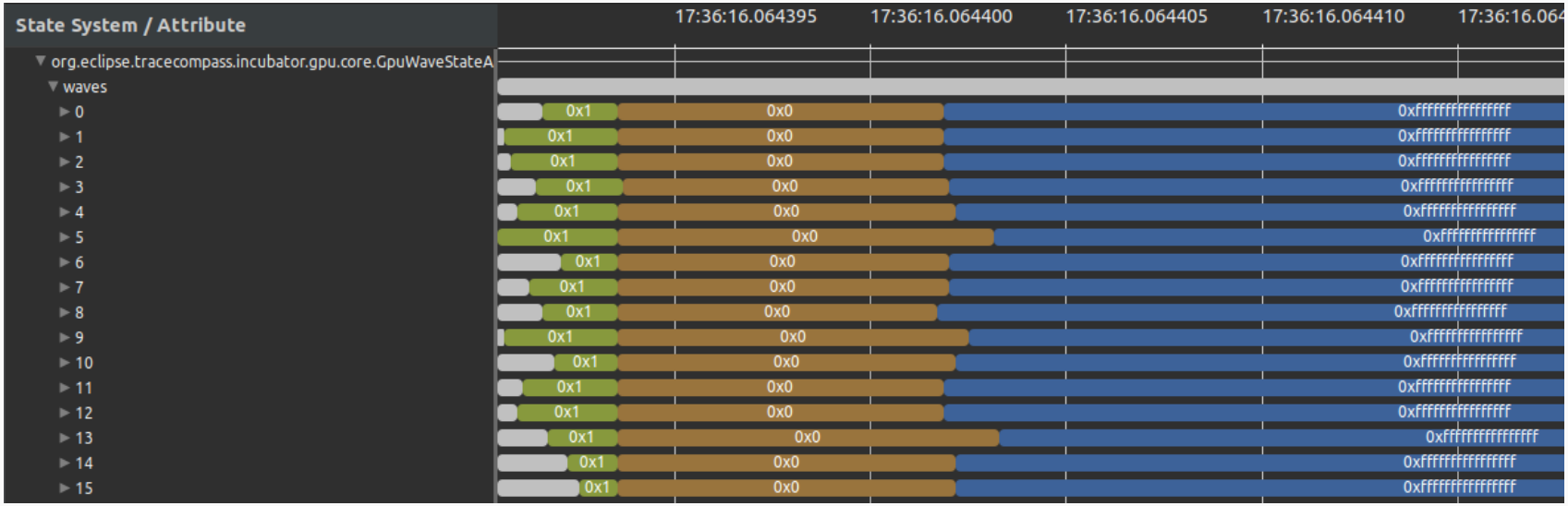
- CFG counters can generate the total number of FLOPs
- Original run allows us to compute the Arithmetic Intensity (FLOPs/s)
  - A quick roofline plot shows we're way bellow expected performance
- We decide to collect more data for analysis with the event collection pass
  - Precise thread divergence
  - If needed, obtain accessed addresses for locality analysis

Implementation - Results

- Main design goals are runtime overhead and ease of use
  - Only a few compiler flags
- First results are encouraging : overhead of $\sim 7\times$ to $15\times$
  - Lower than comparable work
- Trade-off between runtime and memory overhead
- Trace format, analysis and graphs implemented in a TraceCompass plugin

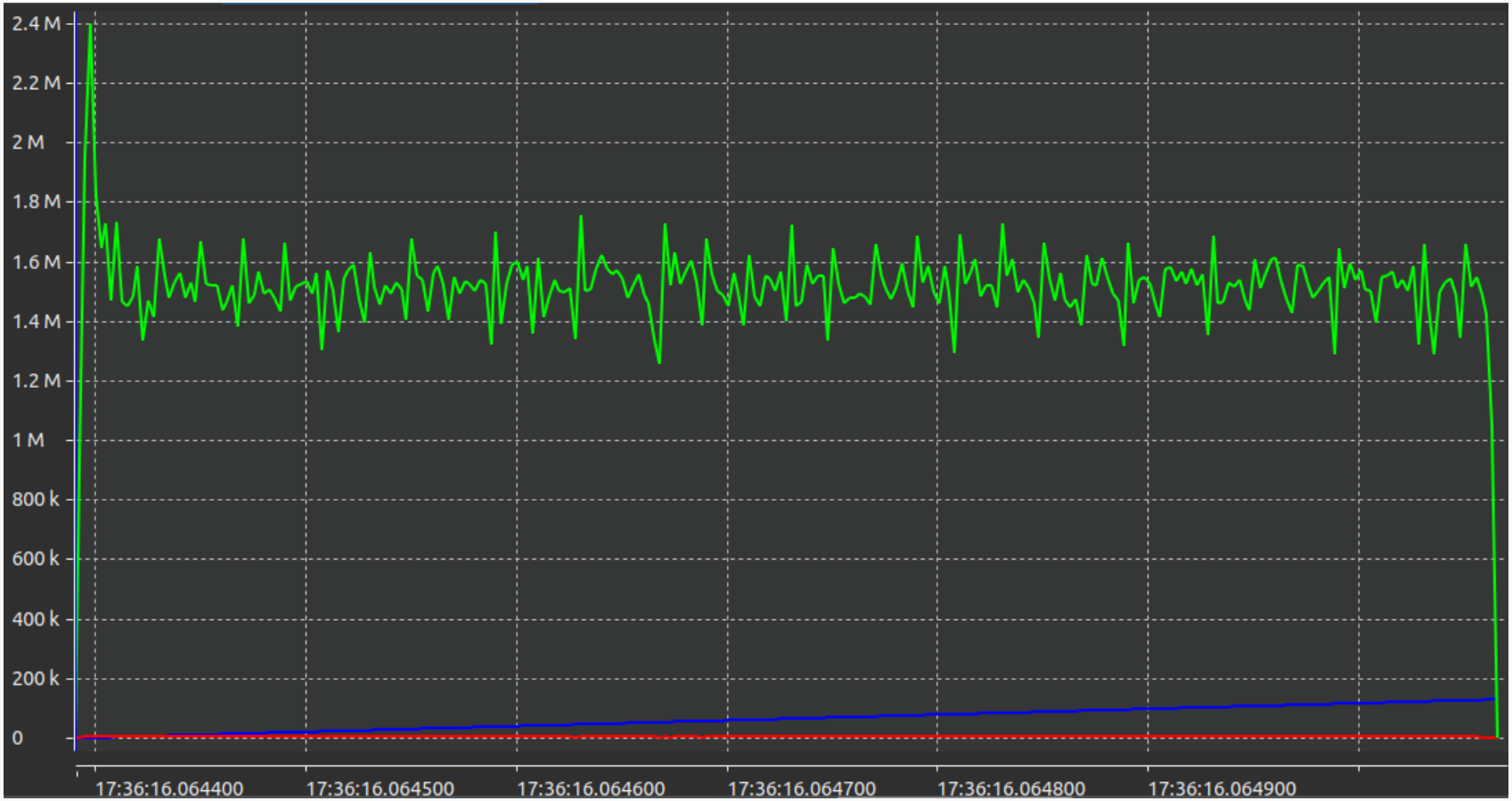
Following screenshots taken from the TraceCompassGpu plugin

State system analysis



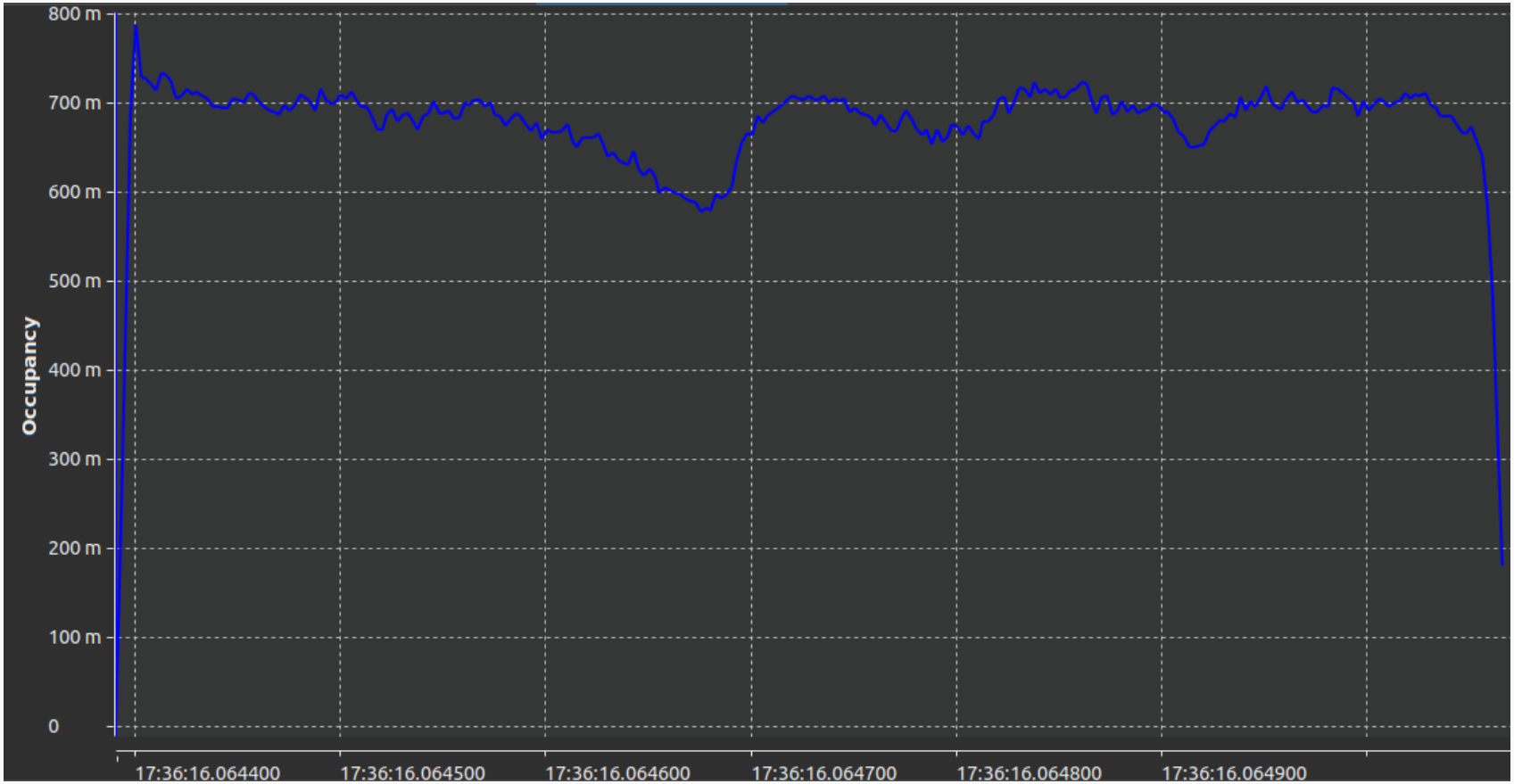
Which basic block each wavefront is executing

Cumulated waves througput



In green, total throughput (FLOP/s) of the kernel

Occupancy



Instantaneous occupancy of the device (hovering around 70%)

Conclusion and future work

- Next step is implementing a runtime event collector on the GPU
  - would eliminate the need for the first CFG run
  - a challenge for GPU execution models
- Integrate well-known performance models and microarchitectural particularities in the analysis framework
- Available freely on Github, feedback and/or use cases are more than welcome

Sebdar/hip-analyzer

Hipcc plugin for performance analysis of HIP applications

C++ 1

Sebdar/TraceCompassGpu

Trace Compass GPU plugins

Java